PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
M.I.T.
CAMBRIDGE, MASSACHUSETTS 02139

PDP-37

PROGRAMMED QUEUES

in the PDP-1-X

June 7, 1966

PROGRAMMED QUEUES in the PDP-1-X

One of the problems which will arise in programming for the PDP-1-X is the allocation of memory (for use as private data storage) to an unpredictable number of processes, within a single program. For example, this problem can occur in any program which has created an inferior computation which has the privilege of forking. If the inferior sphere produces fifty faulty processes, the superior program must be able to absorb fifty new processes at its "fault address" and supply them with private data storage as required. But if the superior program only has twenty blocks of available space, it's in trouble.

(In a segmented computer system, this is not a problem since each new process could create for itself a new segment for storage of private data. Unfortunately, the PDP-1-X is not segmented.)

To help solve this problem, the PDP-1-X will provide three new meta-instructions: create queue, enter queue, and release queue.

The meta-instruction

i:= create queue t;

creates an owned queue capability with C-list index i and threshold t. (t is an integer.)

A process is said to have been through the queue if it has executed an enter queue i; has resumed execution, and has not yet executed a release queue i;.

When a process executes an enter queue i; it will be allowed to continue only if less than t processes have been through the queue. Otherwise, it will be placed in a waiting state and will remain waiting until a sufficient number of processes have released the queue to let all processes ahead of it, and it, through the queue.

The following is a pseudo-program which shares a single drum buffer among any number of processes. The reason queueing is used here is that the data transfer time (with wait time) is longer than the lock interval. This method could be used to copy a file, as fast as possible, without regard to the order (in time) in which blocks are copied, by having one process per block to be copied.

```
.
.
.
i:= create queue 1;
.
.
.



.
.
.
verify page (make sure it's on the drum, not dectape);
enter queue i;
read page;
.
.  } compute, maybe write the page back or someplace else
.
release queue i;
.
.
.
```